PATENT ABSTRACTS OF JAPAN

(11)Publication number : 05-324353
(43)Date of publication of application : 07.12.1993

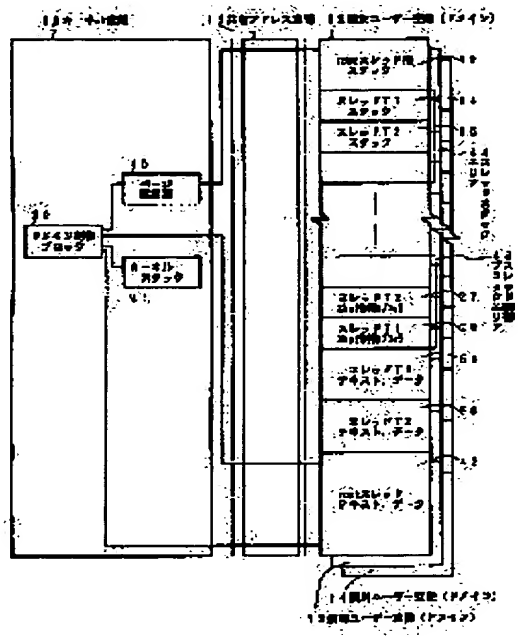
(51)Int.Cl. G06F 9/46

(21)Application number : 04-151449
(22)Date of filing : 19.05.1992

(71)Applicant : NIPPON TELEGR & TELEPH CORP <NTT>
(72)Inventor : TANAKA SATOSHI
KUBOTA MINORU
MARUYAMA KATSUMI

(54) SYSTEM FOR CONTROLLING LIGHT-WEIGHT PROCESS

(57)Abstract:
PURPOSE: To evade the overhead of process control by means of change-over to a priviledge mode by arranging a process control block in space which can be accessed by means of the non-priviledge mode of logical address space.
CONSTITUTION: Individual user space 12 is address space which can be accessed by a user mode and has a high-order 2G byte address, where an application program is operated. Shared address space 11 is the only one space in a computer system having address which continues from whole individual user space 12 and kernel space 10, which can be accessed at the time of the user mode as it is. A process control structure body where information for managing and controlling the execution of light-weight process is held is arranged in logical address space which can be accessed by the non-priviledge mode of logical address space, a stack being required for execution is adopted as the one only for the non-priviledge mode and plural kinds of light-weight process are executed inside the logical address in parallel.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

<Partial Translation of Japanese Unexamined Patent Publication No. H05-324353>

[0018] Fig. 2 shows a constitutional example of a logical address space.

[0019] A reference number 10 represents the kernel space having the size of one gigabyte. One can access the kernel space only when the processor is in a state of the kernel mode.

[0020] The reference numbers 11 and 12 represent a shared address space and an individual user space, respectively.

[0021] One can access the individual user space 12 only in the user mode. The individual user space has an address of high two gigabytes, in which an application program operates.

[0022] Multiple individual user spaces 12 may be provided.

[0023] The shared address space is an only address on the calculator system. It has a contiguous address from the respective individual user spaces and the kernel space. One can access the shared address space in the user mode.

[0024] The shared address space is used for arranging a message content of the interprocess message communications.

[0025] The individual user space 12 will be hereinafter referred to as a domain, and a lightweight process that operates on the domain will be referred to as a thread.

[0026] The domain and multiple threads that operates on the domain will be explained with reference to Fig. 3.

[0027] Domain control blocks 20 and 21 manages the domains 12 and 13.

[0028] In the domain control block, there are stored an address of a kernel stack in the domain, a page conversion table for constituting the logical address space, and a message reception matrix for message communications among threads, which will be explained below. A priority order for scheduling the domains, information such as runtime and the like, and resource information allocated in the units of domains is also stored in the domain control block.

[0029] Thread control blocks 26 and 27 in the domain 12 manage threads 22 and 23, which operate in the domain 12.

[0030] Thread control blocks 28 and 29 in the domain 13 manage threads 24 and 25, which operate in the domain 13.

[0031] The thread control blocks have context information of the thread, a text of the thread, data, information of the address of the stack, and a unique thread identifier (referred to as a thread ID).

[0032] Root threads 30 and 31 having a thread control function operates for controlling the threads in the respective domains.

[0033] The root threads 30 and 31 are generated by the kernel when the domains are generated. They initialize the domains.

[0034] The root threads performs the thread control function in the domain, such as the generation and deletion of the thread, the switch of the threads in the domain, etc.

[0035] The constitution of the domain and thread will be explained with reference to Fig. 1.

[0036] The domain control block 20 has a dedicated kernel stack 41 for the domain, the page conversion table 40, and the address information to the root thread 42.

[0037] In the domain 12 managed by the page conversion table 40, a stack area 45 for the root thread is provided

in the upper address. The text/data 42 for the root thread, the text/data 51 for the thread T1, and the text/data 50 for the thread T2 are provided in the lower-order address.

[0038] When generating the domain, a thread control block area 43 for the purpose of the thread control block is provided for pages for a certain number of thread control blocks is provided.

[0039] Furthermore, the stack area 45 for the root thread is follow by a certain size of the stack area 44 for the thread is provided.

[0040] The root thread having the domain process control function manages these areas.

[0041] The vacancy state of these areas, which are provided when the domain is generated, is managed, and if there are sufficient free area, they are reused.

[0042] On the contrary, if the size provided when the domain is generated exceeds, a memory space is provided by the kernel, and the page conversion table 40 is rewritten. Thus, these areas are expanded.

[0043] Thus, the dedicated kernel stack 41 for the domain and the page conversion table 40 are allocated to the domain 12, and the thread control block area 43, and then the stack area 44 for the thread are provided. Then, the control is given to the root thread 42.

[0044] The root thread 42 initializes the thread control block area 43 and the stack area 44 for the thread, and then the control is given to the application program. Then, the thread control blocks 26 and 27 for the threads T1 22 and T2 23 in the domain as well as the thread stacks 54 and 55 are allocated. Thus, the thread is generated.

[0045] Fig. 5 shows a processing flow for generating the domain.

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号

特開平5－324353

(43)公開日　平成5年(1993)12月7日

(51)Int.Cl.[5]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　9/46	３４０　Ｂ	8120－5B		

審査請求　未請求　請求項の数1（全 12 頁）

(21)出願番号　特願平4－151449

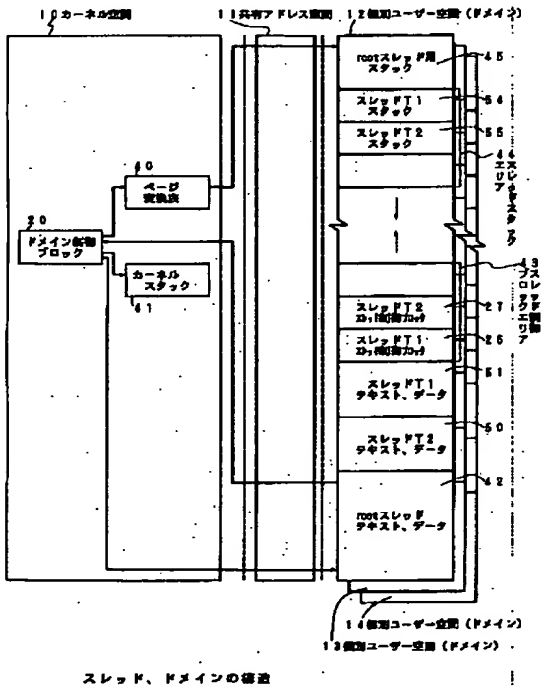
(22)出願日　平成4年(1992)5月19日

特許法第30条第1項適用申請有り　1991年11月21日　社団法人電子情報通信学会発行の「電子情報通信学会技術研究報告Ｖｏｌ．91　Ｎｏ．333」に発表

(71)出願人　000004226
日本電信電話株式会社
東京都千代田区内幸町一丁目1番6号
(72)発明者　田中　聡
東京都千代田区内幸町一丁目1番6号　日本電信電話株式会社内
(72)発明者　久保田　稔
東京都千代田区内幸町一丁目1番6号　日本電信電話株式会社内
(72)発明者　丸山　勝己
東京都千代田区内幸町一丁目1番6号　日本電信電話株式会社内
(74)代理人　弁理士　玉蟲　久五郎

(54)【発明の名称】　軽量プロセス制御方式

(57)【要約】
【目的】　必要メモリ量を小さくするとともにプロセス制御のオーバーヘッドを避けることのできる軽量プロセス制御方式を提供する。
【構成】　実行モードとして、少なくとも特権モード（カーネルモード）と非特権モード（ユーザーモード）の状態を持ち、複数の論理アドレス空間を制御する機能を持つプロセッサから構成される計算機システムのプロセスを軽量化した軽量プロセス制御方式において、軽量プロセスの実行を管理、制御するための情報を保持するプロセス制御構造体を論理アドレス空間の非特権モードでアクセス可能な論理アドレス空間に配置し、実行に必要なスタックを非特権モード用だけとし、論理アドレス内で複数の軽量プロセスが並行実行する。同時に多数のプロセスがプロセッサ上で動作し、短時間に、多数のプロセスが生成、消滅を繰り返し、複数のプロセスが互いに情報をプロセス間通信しながら並行多重処理を進めていくアプリケーションプログラムに適用する、オーバーヘッドの小さいプロセス制御方式を構成する。



スレッド、ドメインの構造

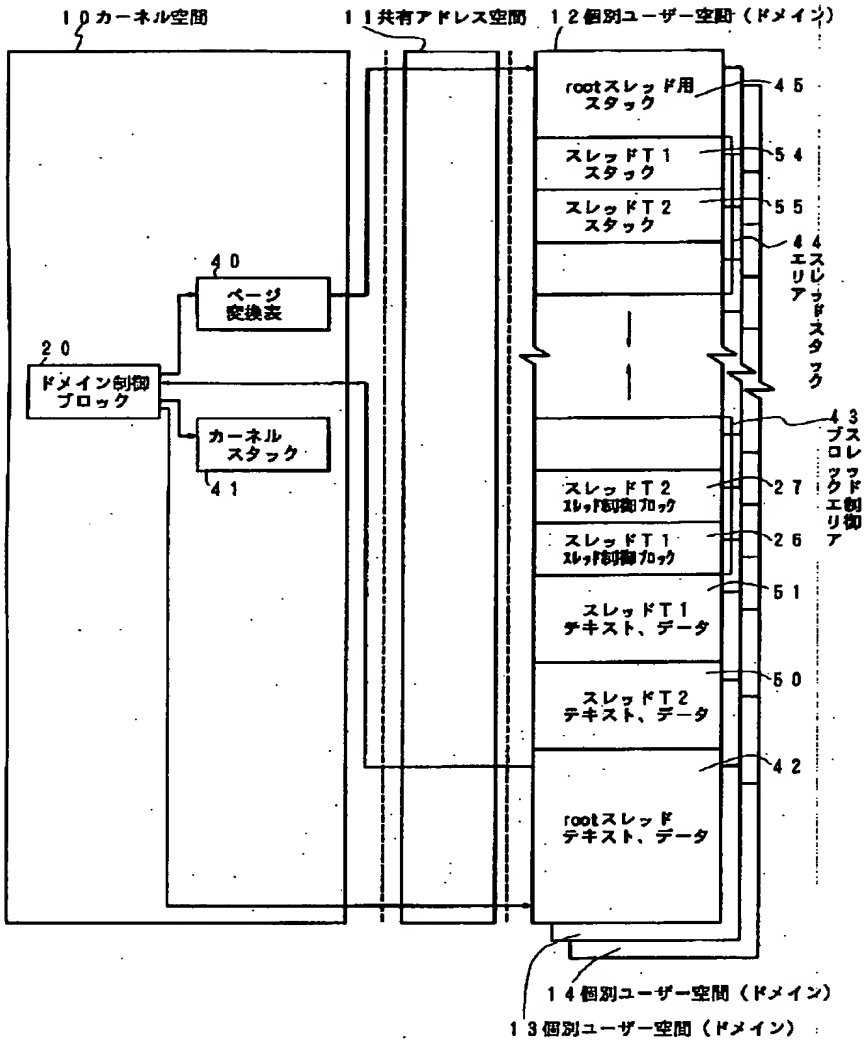
【特許請求の範囲】
【請求項１】　実行モードとして、少なくとも特権モードと非特権モードの状態を持ち、複数の論理アドレス空間を制御する機能を持つプロセッサから構成される計算機システムのプロセスを軽量化した軽量プロセス制御方式において、
前記軽量プロセスの実行を管理、制御するための情報を保持するプロセス制御構造体を論理アドレス空間の非特権モードでアクセス可能な論理アドレス空間に配置し、実行に必要なスタックを非特権モード用だけとし、論理アドレス内で複数の軽量プロセスが並行実行することを特徴とする、軽量プロセス制御方式。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】本発明は、実行モードとして、少なくとも特権モードと非特権モードの状態を持ち、複数の論理アドレス空間を制御する機能を持つプロセッサから構成される計算機システムにおけるプロセスを軽量化した軽量プロセス制御方式に関し、特に同時に多数のプロセスがプロセッサ上で動作し、短時間に、多数のプロセスが生成、消滅を繰り返し、複数のプロセスが互いに情報をプロセス間通信しながら並行多重処理を進めていくアプリケーションプログラムに適用する、オーバーヘッドの小さいプロセス制御方式に関するものである。
【０００２】
【従来の技術】従来使用されているオペレーティングシステムとして、ＵＮＩＸの場合について、図７を用いて説明する。
【０００３】図７は、ＵＮＩＸにおけるプロセスの構造を示している。
【０００４】ＵＮＩＸの１プロセスは一つの論理アドレス空間に対応しており、論理アドレス空間８８、８９、９０において、それぞれ一つのプロセスが動作する。
【０００５】一つのプロセスに対応し、カーネル空間８２内のプロセス構造体８０がアロケートされ、スケジューリング情報、識別子、メモリなどの資源がＵＮＩＸカーネルによって管理される。
【０００６】そして、ページ変換表８１によってプロセスの動作する論理アドレス空間８８が管理される。
【０００７】論理アドレス空間８８内の上位アドレスに、プロセスのシステムコール実行用のカーネルスタック８７、プロセスのコンテキストを管理するためのｕ構造体８６、アドレス空間内の下位アドレスに、プロセスのテキスト８３とデータ８４が配置される。
【０００８】このように、ＵＮＩＸのプロセスは、それを構成するためのリソースが多く、それらを確保したり、あるいは解放するためのオーバーヘッドは非常に大きい。
【０００９】そのため、ＵＮＩＸのプロセスの生成と消滅、切り替えのオーバーヘッドが大きい。
【００１０】特に、同時に多数の呼を実時間に処理する必要のある交換処理をプロセスで構成した場合、プロセスは比較的小規模だが、その数が多い、１、２Ｋステップ走行するごとにメッセージ通信の事象待ちを生じ、コンテキストスイッチが頻繁に発生する、という特徴がある。
【００１１】
【発明が解決しようとする課題】上記のような交換処理に従来のＵＮＩＸプロセスを適用した場合、プロセスの生成、消滅、コンテキストスイッチのオーバーヘッドのため、高い多重処理能力と実時間処理能力を同時に満足するようなシステムの構築は困難であり、必要なメモリ量も増大するという問題がある。
【００１２】本発明の目的は従来の問題点を解決し、必要メモリ量を小さくするとともにプロセス制御のオーバーヘッドを避けることのできる軽量プロセス制御方式を提供することにある。
【００１３】
【課題を解決するための手段】本発明は上記目的を達成するため、実行モードとして、少なくとも特権モードと非特権モードの状態を持ち、複数の論理アドレス空間を制御する機能を持つプロセッサから構成される計算機システムのプロセスを軽量化した軽量システム制御方式において、前記軽量プロセスの実行を管理、制御するための情報を保持するプロセス制御構造体を論理アドレス空間の非特権モードでアクセス可能な論理アドレス空間に配置し、実行に必要なスタックを非特権モード用だけとし、論理アドレス内で複数の軽量プロセスが並行実行することを特徴とする。
【００１４】
【作用】本発明は、同時に多数のプロセスがプロセッサ上で動作し、短時間に、多数のプロセスが生成、消滅を繰り返し、複数のプロセスが互いに情報をプロセス間通信しながら並行多重処理を進めていくアプリケーションプログラムに適用する、オーバーヘッドの小さいプロセス制御方式で、プロセスの実行管理、制御するための情報を保持するプロセス制御構造体を論理アドレス空間の非特権モードでアクセスできる空間に配置し、プロセスは非特権モード用のスタック一本のみを持つ、論理アドレス空間内で並行実行可能な複数の軽量プロセスを構成する。
【００１５】
【実施例】以下、本発明の実施例を図１を用いて説明する。
【００１６】本実施例では、プロセッサは３２bit 幅のアドレスを取り扱い、プロセッサの実行モードとして、特権モードであるカーネルモードと、非特権モードであるユーザーモードを持ち、ページ変換表のアドレスをプロセッサの制御レジスタに設定しておくことにより、論理アドレスから物理アドレスへの変換を自動的に行なう

機能を持っているものとする。
【００１７】アプリケーションとして、交換処理を考える。
【００１８】図２に論理アドレス空間の構成例を示す。
【００１９】１０はカーネル空間で１Ｇバイトの大きさを持ち、プロセッサがカーネルモードの状態にある時のみアクセス可能な空間である。
【００２０】１１は共有アドレス空間、１２は個別ユーザー空間である。
【００２１】個別ユーザー空間１２は、ユーザーモードでアクセス可能なアドレス空間で、上位２Ｇバイトのアドレスを持ち、アプリケーションプログラムが動作する。
【００２２】個別ユーザー空間１２は複数存在することができる。
【００２３】共有アドレス空間１１は計算機システム上唯一の空間で、すべての個別ユーザー空間、カーネル空間から連続したアドレスを持ち、ユーザーモードのままでアクセス可能である。
【００２４】共有アドレス空間１１は、プロセス間メッセージ通信のメッセージ内容を配置するために使用する。
【００２５】以下、個別ユーザー空間１２をドメイン、ドメインで動作する複数の軽量プロセスをスレッドとよぶ。
【００２６】図３を用いて、ドメインとその上で動作する複数のスレッドについて説明する。
【００２７】ドメイン１２、１３は、カーネル空間１０内のドメイン制御ブロック２０、２１で管理される。
【００２８】ドメイン制御ブロックは、ドメインで持つカーネルスタックのアドレス、論理アドレス空間を構成するためのページ変換表、後述するスレッド間メッセージ通信用のメッセージ受信行列、ドメインのスケジューリングのための優先順位、実行時間などの情報、ドメイン単位でアロケートされる資源情報などが格納される。
【００２９】ドメイン１２の中で動作するスレッド２２、２３は、ドメイン１２内のスレッド制御ブロック２６、２７より管理される。
【００３０】ドメイン１３の中で動作するスレッド２４、２５は、ドメイン１３内のスレッド制御ブロック２８、２９より管理される。
【００３１】スレッド制御ブロックはスレッドのコンテキスト情報、スレッドのテキスト、データ、スタックのアドレスの情報、ユニークに決まるスレッド識別子（スレッドＩＤと呼ぶ。）を持つ。
【００３２】各ドメイン内でのスレッド制御を行うため、スレッド制御機能を持つrootスレッド３０、３１が動作する。
【００３３】このrootスレッド３０、３１は、ドメイン生成時にカーネルにより生成され、ドメインの初期設定を行なう。
【００３４】そして、スレッドの生成、削除、ドメイン内のスレッドの切り替えなどのドメイン内のスレッド制御機能はrootスレッドが行なう。
【００３５】図１を用いて、ドメインとスレッドの構造について説明する。
【００３６】ドメイン制御ブロック２０は、ドメイン専用のカーネルスタック４１、ページ変換表４０、rootスレッド４２へのアドレス情報を持つ。
【００３７】ページ変換表４０で管理されるドメイン１２では、アドレスの上位にrootスレッド用のスタックエリア４５が確保され、下位アドレスにrootスレッドのテキストとデータ４２、スレッドＴ１用のテキストとデータ５１、スレッドＴ２用のテキストとデータ５０が置かれる。
【００３８】ドメイン生成時には、スレッド制御ブロックのためのスレッド制御ブロックエリア４３が、一定個数のスレッド制御ブロック分のページが確保される。
【００３９】また、rootスレッド用のスタックエリア４５の続きには、スレッド用のスタックエリア４４が一定の大きさ分確保される。
【００４０】これらの領域は、ドメイン内プロセス制御機能をもつrootスレッドにより管理される。
【００４１】ドメイン生成時に確保されたこれらの領域は、空塞状態が管理され、十分な空き領域がある場合には再利用される。
【００４２】しかし、ドメイン生成時に確保された大きさを越えた場合には、カーネルによりメモリ空間が確保され、ページ変換表４０が書き換えられ、これらのエリアは拡張される。
【００４３】ドメイン制御ブロック２０、ドメイン専用のカーネルスタック４１、ページ変換表４０がドメイン１２用にアロケートされ、スレッド制御ブロックエリア４３、スレッド用のスタックエリア４４が確保されたあと、rootスレッド４２に制御が渡される。
【００４４】rootスレッド４２はスレッド制御ブロックエリア４３、スレッド用のスタックエリア４４の初期設定を行ない、アプリケーションプログラムに制御が渡り、ドメイン内のスレッドＴ１　２２、Ｔ２　２３のスレッド制御ブロック２６、２７、および、スレッドのスタック５４、５５がアロケートされ、スレッドが生成される。
【００４５】図５に上記のドメイン生成時の処理フローを示す。
【００４６】本実施例でのスレッド間メッセージ通信について説明する。
【００４７】ここでは、交換処理に適用できるように、スループットが高く、遅延時間の短いメッセージ通信機構を実現するために、通信するデータを共有空間上におき、そこへのポインターをスレッド間で通信する。

【００４８】共有空間上の通信データをメッセージバッファと呼ぶ。
【００４９】メッセージを送信しようとするスレッドは、共有空間上のメッセージバッファを捕捉し、そこにメッセージの内容を書き込む。
【００５０】受信スレッドのスレッドＩＤを指定して、そのメッセージを送信する。
【００５１】送信後は、送信スレッドはメッセージバッファに対してアクセスすることができない。
【００５２】スレッドはスレッド制御ブロックにメッセージの受信キューを持っていて、スレッドに到着したメッセージは、到着順に受信キューにキューイングされる。
【００５３】受信スレッドは、自分の受信キューから到着しているメッセージバッファを一つ選択して受信する。
【００５４】ここでは、通信網内の１交換システムをノードと呼び、同じノード内、あるいは異なるノードに存在する二つのスレッド間でメッセージを通信して交換処理を行なう場合を例にして説明する。
【００５５】ノード間の通信はカーネル内にあるノード間通信制御機能が行なうものとする。
【００５６】図４で、本実施例でのメッセージ通信処理について具体的に説明する。
【００５７】メッセージ通信処理は受信スレッドの存在する位置に応じて、以下の三つの場合がある。
【００５８】1.受信スレッドが同じドメイン内に存在する場合。
【００５９】スレッド２２からスレッド２３へ、メッセージバッファ６０を送信する場合を説明する。
【００６０】スレッド２２はメッセージバッファ６０をアロケートし、通信内容を書き込む。
【００６１】メッセージ送信では、スレッド２３のスレッド制御ブロック２７のメッセージ受信キューにメッセージバッファ６０がキューイングされる。
【００６２】2.受信スレッドが同一ノード内の別ドメインに存在する場合。
【００６３】スレッド２２からスレッド２４へ、メッセージバッファ６１を送信する場合を説明する。
【００６４】スレッド２２がメッセージバッファ６１を送信する時、スレッド制御ブロック２８に直接メッセージをキューイングできないので、スレッド２４の存在するドメイン１３のドメイン制御ブロック２１にメッセージをキューイングする。
【００６５】ドメイン１３がプロセッサの使用権を獲得して、ドメイン１３に制御が切り替わる時に、ドメインに到着しているメッセージがドメイン１３内のスレッド２４のスレッド制御ブロック２８にキューイングされる。
【００６６】3.受信スレッドが別ノードに存在する場合。
【００６７】スレッド２２からスレッド４へメッセージバッファ６２を送信する場合を説明する。
【００６８】受信スレッドＩＤから、受信スレッドは別ノードに存在することがわかるので、自分のノードのノード間通信制御機能に、メッセージの通信を依頼して、リターンする。
【００６９】送信側ノード間通信制御機能はメッセージバッファ６２の内容を受信ノードに送信する。
【００７０】受信側ノード間通信制御機能はメッセージバッファ６３を確保し、送信側ノード間通信制御機能から送られてきたメッセージをそこに格納する。
【００７１】そして、受信側ノード間通信制御機能は受信スレッド４の存在するドメイン７のドメイン制御ブロック６にメッセージをキューイングする。
【００７２】ドメイン制御ブロック６にキューイングされたメッセージバッファ６３は、ドメイン７にプロセッサの実行が移される時に、受信スレッド４のスレッド制御ブロック５にキューイングされる。
【００７３】図６に、一例としてメッセージ送信の場合の処理フローを示す。
【００７４】メッセージ受信キューは、スレッド制御ブロックあるいはドメイン制御ブロックを受信キューの先頭とし、メッセージバッファの双方向リンクを組むことによって、構成される。
【００７５】特に、スレッド制御ブロックのメッセージ受信キューのアクセスには排他制御を行なう。
【００７６】先のメッセージ通信処理で示したように、別ノード上のスレッドにメッセージを送信する場合、ノード間メッセージ通信機能にメッセージ送信を依頼した後で、スレッドには制御が戻される。
【００７７】その後、受信スレッドが存在しないなどの通信エラーが発生した場合、そのエラーの発生を非同期に知らせ、優先的に処理する必要がある。
【００７８】そのため、非同期に発生するエラーはメッセージとしてスレッドに通知される。
【００７９】このメッセージを緊急メッセージと呼ぶ。
【００８０】緊急メッセージは、通常のメッセージの受信キューの先頭にキューイングされ、優先的に処理される。
【００８１】スレッドのメモリエラーなど、エラーを発生したスレッドが引続き実行することが不可能なエラーの場合、緊急メッセージはrootスレッドに送信される。
【００８２】rootスレッドは、ドメイン生成時には、ドメイン内スレッド制御機能の初期設定をおこない、その後は、次のような処理を繰り返すスレッドとして、実現される。
【００８３】1.rootスレッドに到着している緊急メッセージがあれば、それを処理する。
【００８４】2.ドメインに到着しているメッセージを受

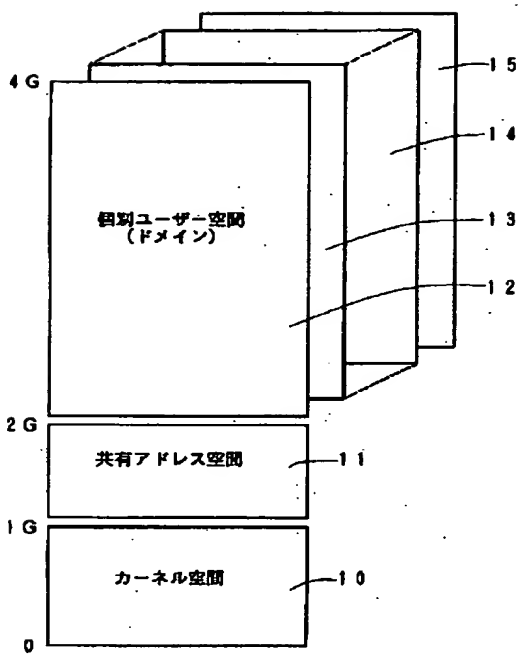
信スレッド制御ブロックにキューイングする。
【００８５】3.次に制御を移すスレッドを選択する。
【００８６】4.メッセージ受信待ちの状態に遷移し、選択されたスレッドに制御をうつす。
【００８７】
【発明の効果】以上述べたように本発明は、プロセス制御ブロックを論理アドレス空間の非特権モードでアクセスできる空間に配置するため、プロセスの生成、消滅は、特権モードで実行する必要がないので、特権モードへの切り替えによる、プロセス制御のオーバーヘッドを避けることができる。
【００８８】またプロセスの実行に必要なスタック、プロセス制御ブロックは、すべて非特権モードでアクセスできる空間にあるため、その確保、解放のオーバーヘッドを小さくすることができる。
【００８９】さらに一つのプロセスを構成するためのカーネルのリソースが少なくできるので、同時に多数のプロセスを制御する必要のある交換システムに適用した場合、必要メモリ量を小さくすることができる。
【００９０】また同じ論理アドレス空間に属するプロセス間でのコンテキストスイッチ、プロセス間メッセージ通信において、特権モードに遷移する必要がないので、これらのオーバーヘッドを小さくできる。
【図面の簡単な説明】
【図１】本実施例におけるドメインとスレッドの構造を示したものである。
【図２】実施例における論理アドレス空間の構成を示した図である。
【図３】実施例におけるドメインとスレッドの制御の関係を示した図である。
【図４】実施例におけるプロセス間メッセージ通信処理の概要を説明する図である。
【図５】図１に対するドメイン生成時の処理フローチャートである。
【図６】図４に対するメッセージ送信の場合の処理フローチャートである。
【図７】従来の技術を説明するための図である。
【符号の説明】
９，１０，８２　カーネル空間
８，１１　共有アドレス空間
７，１２，１３，１４，１５　個別ユーザー空間（ドメイン）
６，２０，２１　ドメイン制御ブロック
４，２２，２３，２４，２５　軽量プロセス（スレッド）
５，２６，２７，２８　２９　スレッド制御ブロック
３０，３１，４２　rootスレッド
６０，６１，６２，６３　メッセージバッファ
４０，８１　ページ変換表
４１，８７　カーネルスタック
４３　スレッド制御ブロックエリア
４４　スレッドスタックエリア
４５　rootスレッド用スタック
５０　スレッドＴ２テキスト、データ
５１　スレッドＴ１テキスト、データ
５４　スレッドＴ１スタック
５５　スレッドＴ２スタック
８０　プロセス構造体
８３　テキスト
８４　データ
８５　ユーザースタック
８６　ｕ構造体
８８，８９，９０　論理アドレス空間

【図1】



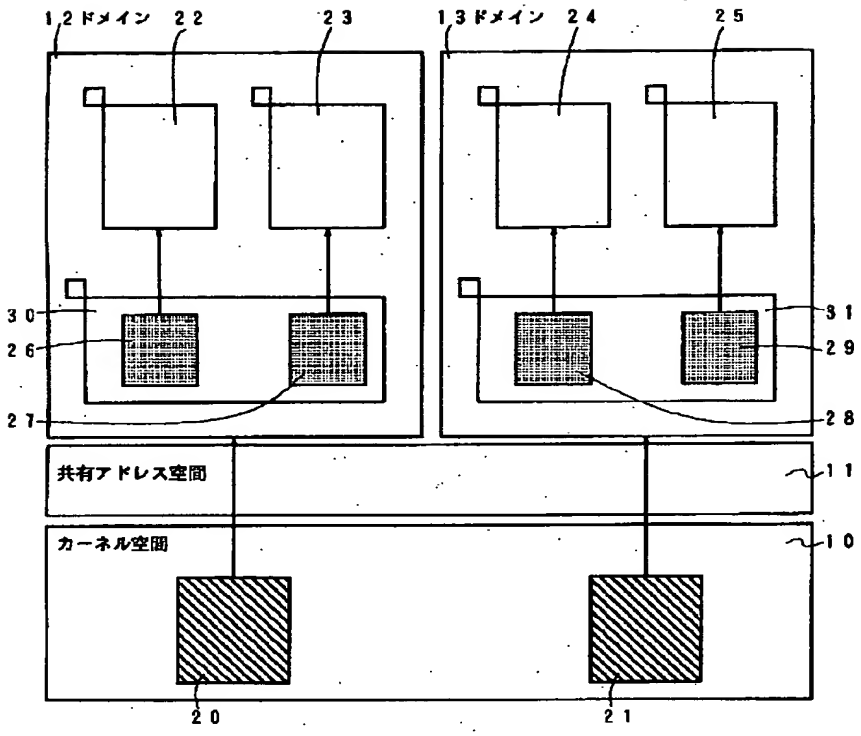
スレッド、ドメインの構造

【図２】



論理アドレス空間の構成

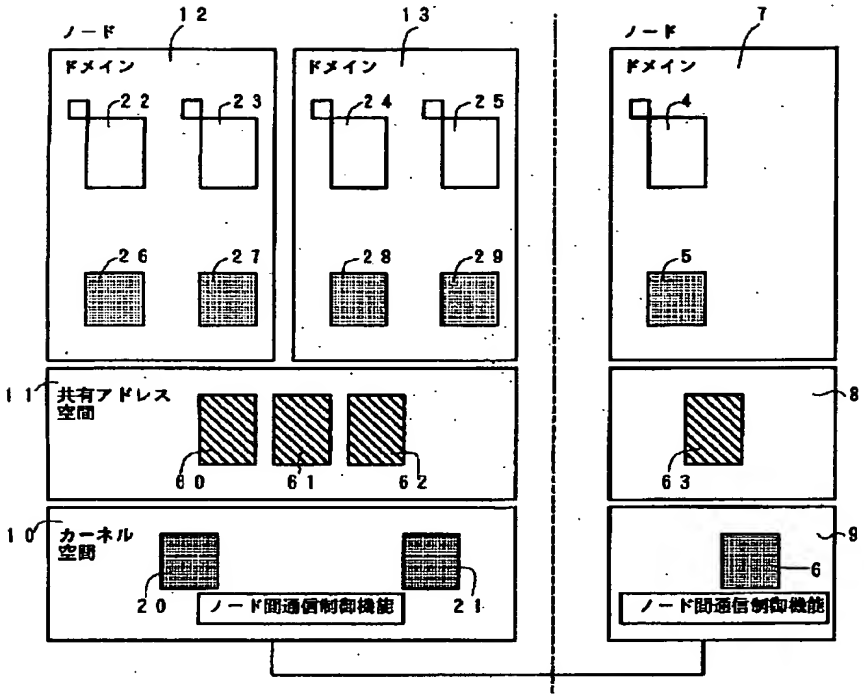
【図3】



20，21…ドメイン制御ブロック
22，23，24，25…スレッド
26，27，28，29…スレッド制御ブロック
30，31…rootスレッド

スレッド、ドメインの制御

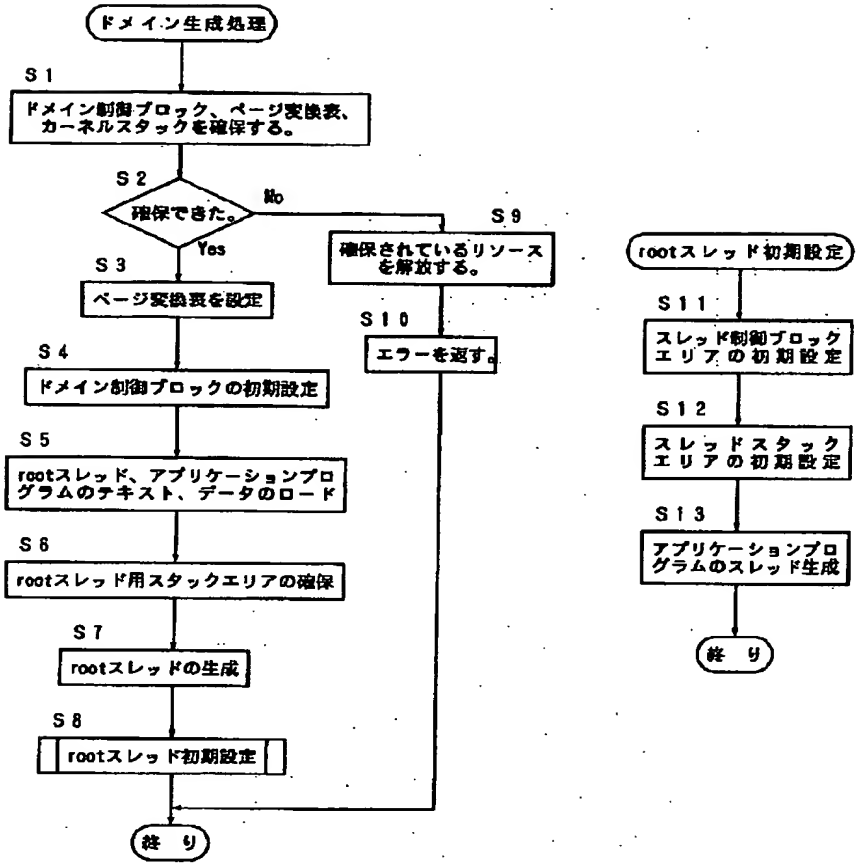
【図4】



6，20，21…ドメイン制御ブロック
4，22，23，24，25…スレッド
5，26，27，28，29…スレッド制御ブロック
60，61，62，63…メッセージバッファ
8…共有アドレス空間
9…カーネル空間

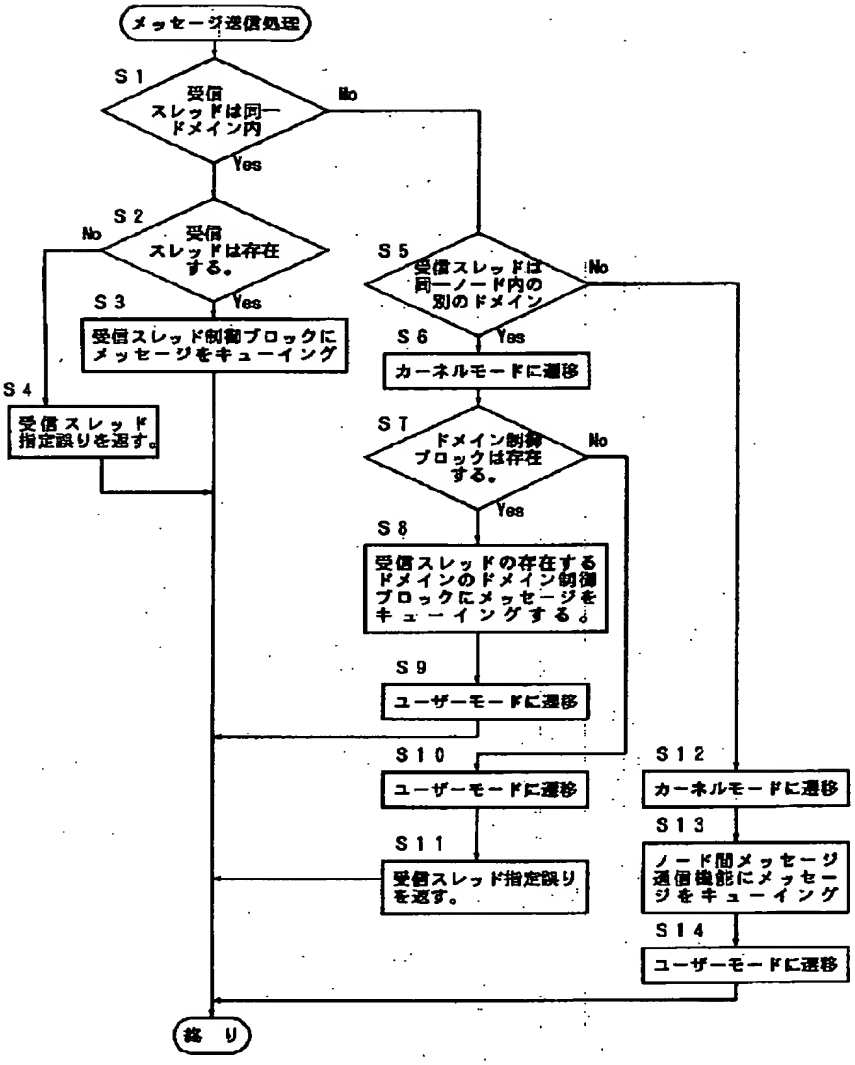
メッセージ通信機能

【図5】



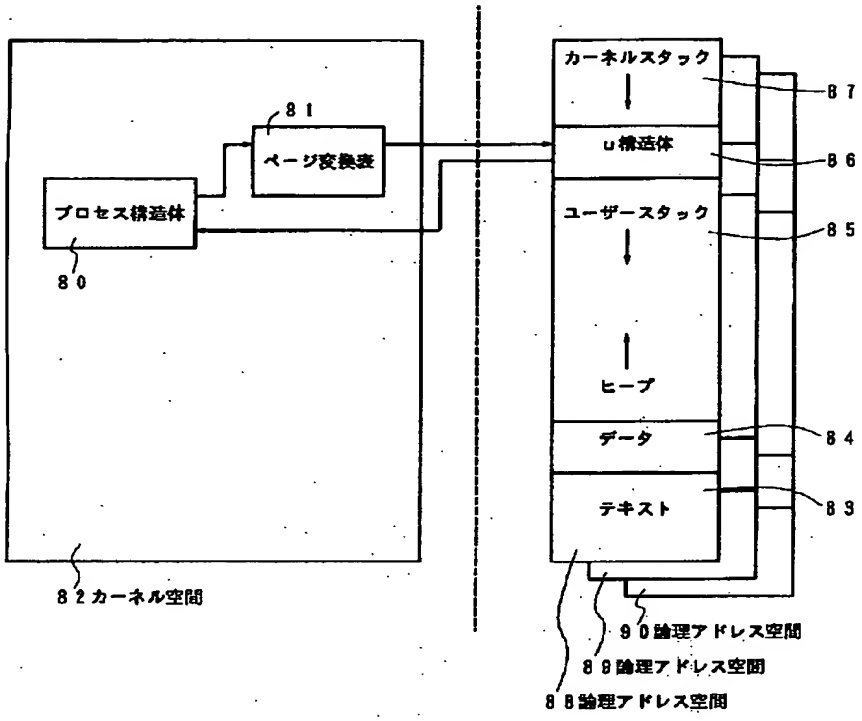
ドメイン生成時の処理フロー

【図6】



メッセージ送信処理フロー

【図7】



UNIXプロセスの構造